An Approach to Generalized Least Significant Bit Steganography

Philipp Harland

May 2025

Email: piranhafisherman@proton.me

Abstract

Least significant bit (LSB) steganography, as commonly known, operates in $\mathbb{Z}_2^*$, or, binary strings. This allows it to store one byte every 3 pixels, which is rather wasteful. Here, we propose a method for constructing a form of generalized LSBS, or, generalized least significant bit steganography.

Contents

1 Construction **1**

2 Worked Example **2**

3 Detector Algorithm **3**

1 Construction

We have the tuple (S, I, k), where S is a string, I is an image, and $k \in \mathbb{Z}_{2\leq n\leq 26}$. We can, for a general positive integer ≥ 2, perform a similar routine as in regular least significant bit steganography – Let $\mathfrak{D}$ be an element of $\mathbb{Z}_{256}^{3mn}$, where m and n are the width and height respectively of the image, I. We select the first d projections of $\mathfrak{D}$, $\pi_1(\mathfrak{D}), ..., \pi_d(\mathfrak{D})$, where d is the length of the concatenated base-k representations of the individual projections of $A_0Z_{25}(S)$. Now, we let $\mathcal{U} = (\pi_1(\mathfrak{D}) \mod k, ..., \pi_d(\mathfrak{D}) \mod k)$. We then add/subtract from each of the d projections, s.t. the k-its[1] represent S.
For example, if we have a string, $S =$ "chair", and $k = 5$, we first have $a \iff 0, c \iff 2, h \iff 7, i \iff 8, r \iff 17$. We represent a and c as 0_5 and 2_5 simply, as well h, i, and r respectively as $12_5, 13_5$, and 32_5. If we were

[1] Here, a generalized "bit" in base n is called an n-it. Under this scheme, we would call a generalized "bit" in base 18 an 18-it, for example.

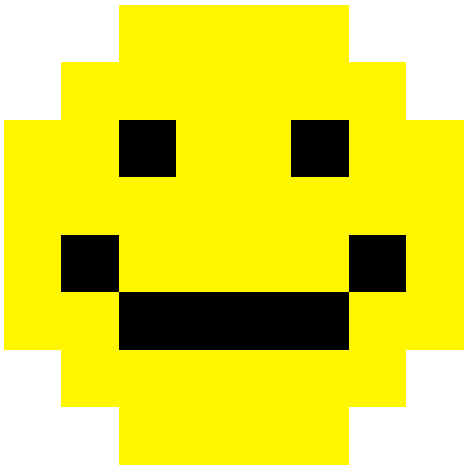

Figure 1: Plain image.

to construct this method similarly to the usual scheme of LSB steganography, i.e. a 128-letter underlying alphabet as in ASCII encoding, we would instead use a single pixel for a single character, since 127 is 3 5-its, and it is equal to 102_5*. This allows for a much higher density of information.*

2 Worked Example

We will use the example of a smiley face in an 8×8 *grid. And, our string will be "smiley", with k=4.*

We will use the first 6 pixels of the image from the top section of the image, i.e. the first 6 elements numbered from left to right, at height 1 in the 8×8 *matrix of* $V_3(\mathbb{Z}_{256})$*-vectors which represents the image.*

We have $s \iff 18, m \iff 12, i \iff 8, l \iff 11, e \iff 4, y \iff 24$*, which are their values in the Vigenère cipher, that is, their alphabetical places minus 1. We have the following base-3 representations of the numbers:* $18_{10} = 200_3, 12_{10} = 110_3, 8_{10} = 122_3, 11_{10} = 102_3, 4_{10} = 11_3, 24_{10} = 220_3$*. We go from R to G to B, and place leading zeros where we do not have 3-its. Fig. 1 is comprised of 3 colors:* $\begin{pmatrix}255\\247\\0\end{pmatrix}$ *(yellow),* $\begin{pmatrix}0\\0\\0\end{pmatrix}$ *(black), and* $\begin{pmatrix}255\\255\\255\end{pmatrix}$ *(white). The vectors,, once taken mod 3, turn out to be* $\begin{pmatrix}0\\1\\0\end{pmatrix}, \begin{pmatrix}0\\0\\0\end{pmatrix}$*, and* $\begin{pmatrix}0\\0\\0\end{pmatrix}$*, respectively. This is because* $255 \cong 0 \mod 3$*,* $247 \cong 1 \mod 3$*, and* $0 \cong 0 \mod 3$*. Our goal is to adjust the first 6 vectors such that they are equal to*

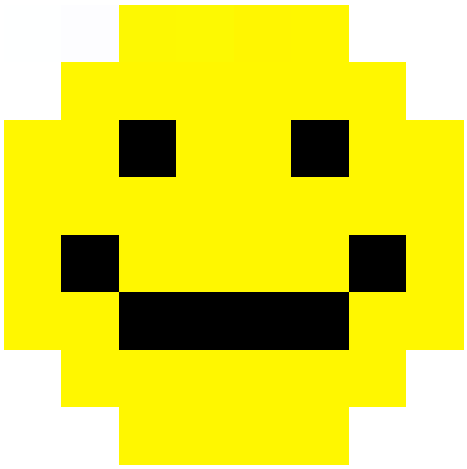

Figure 2: The smiley face with implanted GLSBS.

$\begin{pmatrix}2\\0\\0\end{pmatrix}$, $\begin{pmatrix}1\\1\\0\end{pmatrix}$, $\begin{pmatrix}1\\2\\2\end{pmatrix}$, $\begin{pmatrix}1\\0\\2\end{pmatrix}$, $\begin{pmatrix}0\\1\\1\end{pmatrix}$, *and* $\begin{pmatrix}2\\2\\0\end{pmatrix}$. *One way we can do this, is that we add the vectors* $\begin{pmatrix}2\\0\\0\end{pmatrix}$, $\begin{pmatrix}1\\1\\0\end{pmatrix}$, $\begin{pmatrix}1\\1\\0\end{pmatrix}$, $\begin{pmatrix}1\\0\\2\end{pmatrix}$, $\begin{pmatrix}0\\0\\1\end{pmatrix}$, *and,* $\begin{pmatrix}2\\1\\0\end{pmatrix}$. *However, there are overflows present, so in that case, we instead subtract the appropriate numbers. Applying this, we are left with the following figure: Similarly to in base 2, the changes are barely perceptible to the human eye. However, it may be spotted more easily, which is why this method may be preferred to be used on larger images.*

3 Detector Algorithm

To automatically detect GLSB on an image, I, we can design a program (the following algorithm will be written in pseudocode:

```
1:  select integer k
2:  let ℒ be the set of color channels mod k, enumerated from from
R to G to B, left to right and top to down
3:  const n is the number of k-its used to represent 25
4:  let ℳ be the first nk k-its of ℒ
5:  let every A_j from {A_1, ...A_n} be the jth fixed-length sequence of
k-its.
6:  Let H be the concatenation of mappings from A_j to letters, according
to the sums in base k which the k-its make.
```